GPGPU in Scientific Applications

Wiesław Pietruszkiewicz
West Pomeranian University of Technology

Plan of presentation

- Parallel computing
- GPGPU
- GPGPU technologies
- Scientific applications

Computational limits

- Resources
- Speed:
  - Faster hardware
  - Optimized software
  - Parallelism!

Flynn’s taxonomy

	Single instruction	Multiple instructions
Single data	SISD	MISD
Multiple data	SIMD (GPU)	MIMD

We will focus on the data parallelism (opposite to the task parallelism).

Computational limits

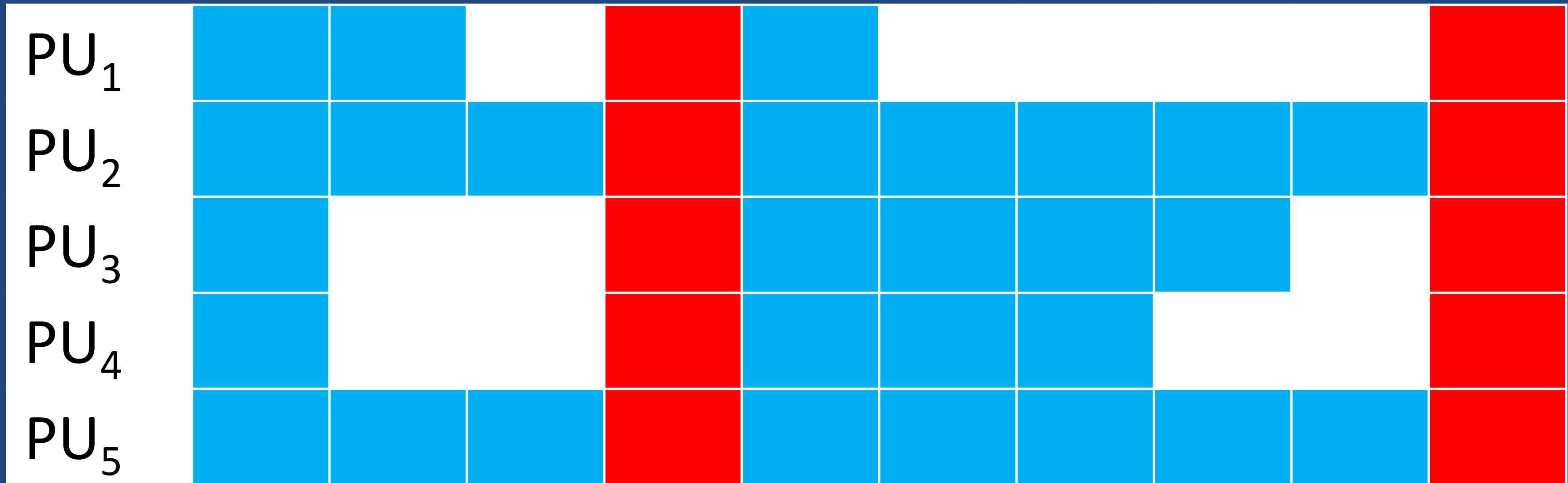
- Task parallelism:
  - Normal on the current OSs run on multi-core processors
  - Independent processes (limited communication)

C P U

- Data parallelism:
  - The same data (vector) to process
  - The elements of vector could be divided and they do not depend one on the another
  - Uses multiple processing units

G P U

Parallel computing



Parallel section

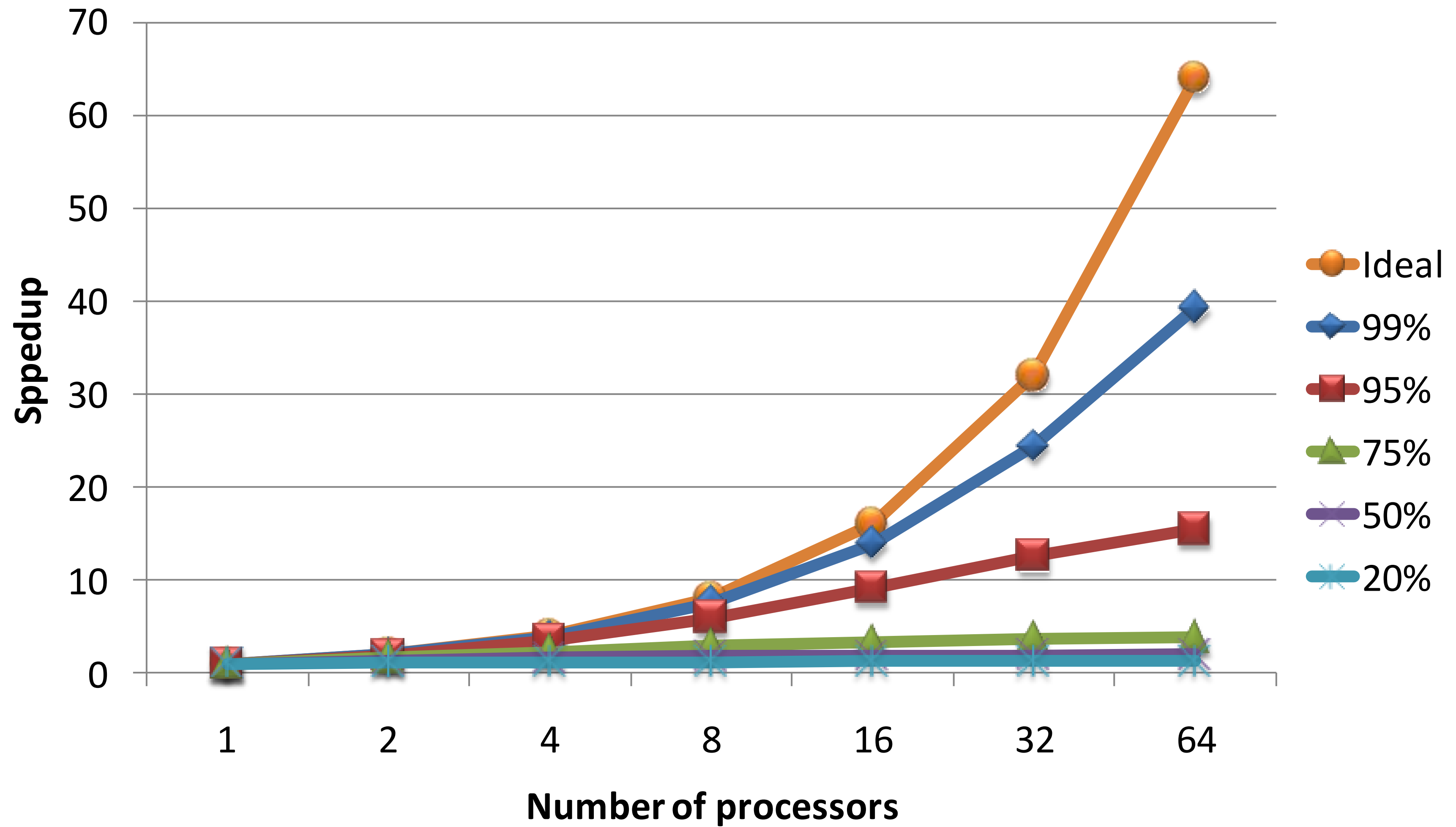
Sequential section

Parallel computing

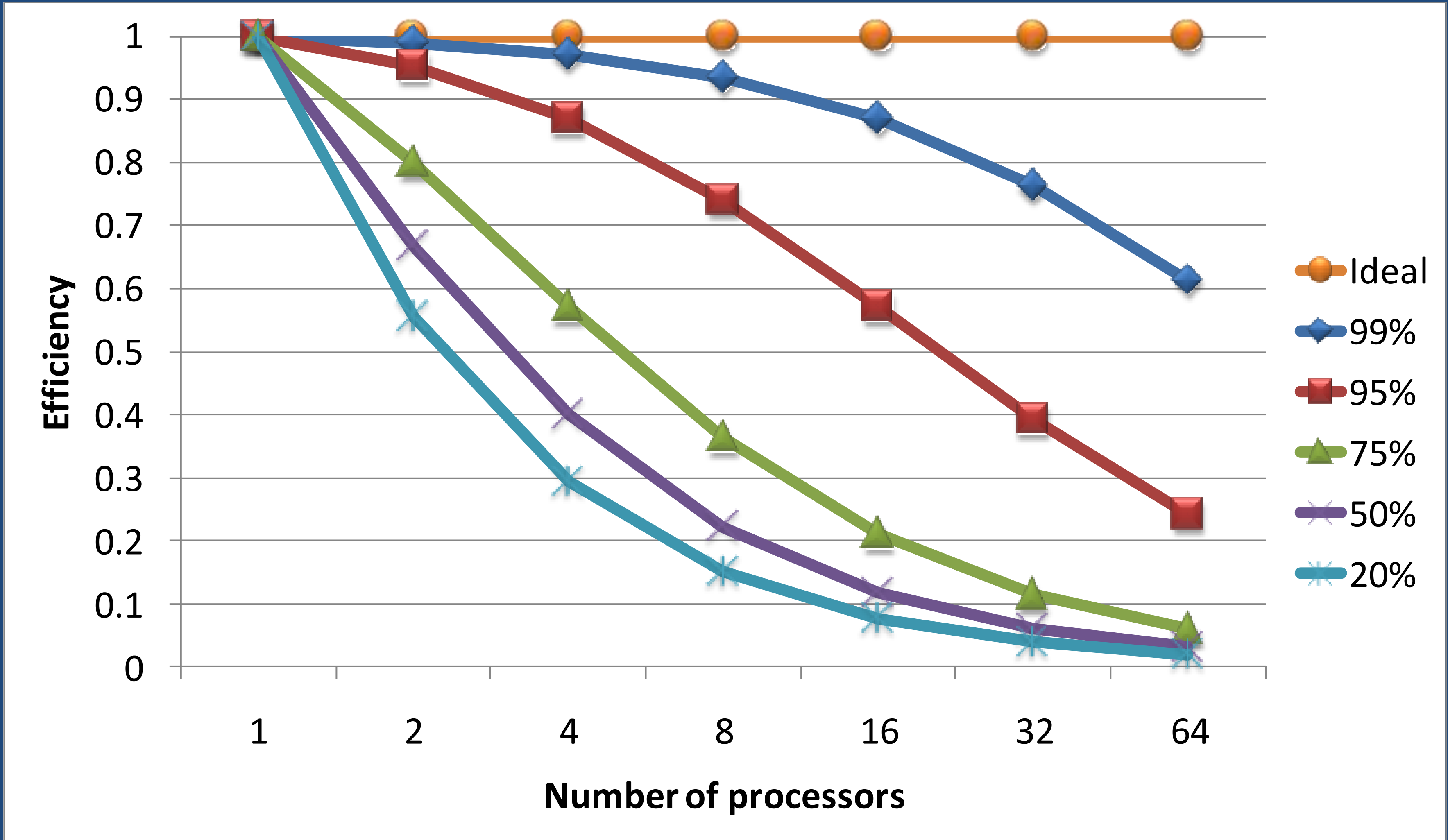
Ahmdal's law – speedup of a programme for the multiple processing units is limited by the time of programme's sequential fraction.

$$Speedup = \frac{1}{(1 - parallel) + \frac{parallel}{processors}}$$

Parallel computing - speedup



Parallel computing - efficiency



Parallel computing

There exists a boundary B that for N processors taking T_N time $T_N \geq B$, regardless what the value of N is.

As B depends on the sequential (non-parallel) part of programme we look for a way to limit this part.

In the result: speedup ↑ efficiency ↑

Parallel computing

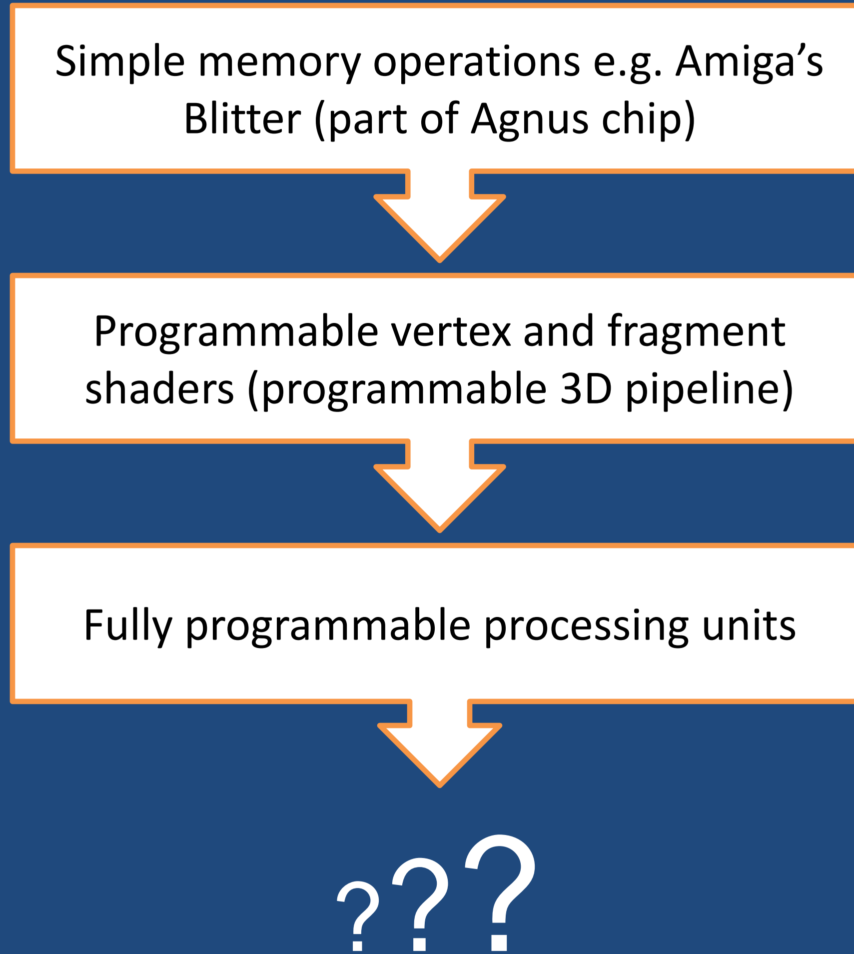
Summarising, we could compute faster by:

- designing better algorithms (not always possible)
- faster machines
- doing taks on several processing units (cores/graphical processors)

The speed of PUs is currently bounded more by quantum physics than by engineering.

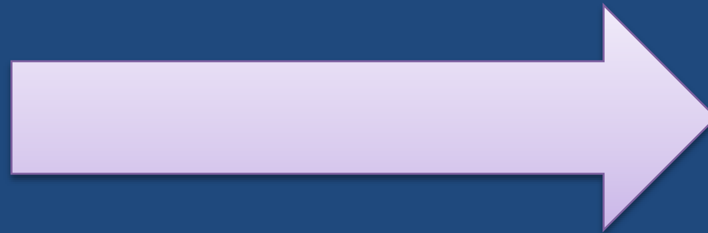
GPGPU

GPU History (a very brief)



Transformation

Graphical
Processing
Unit



General
Purpose
Graphical
Processing
Unit

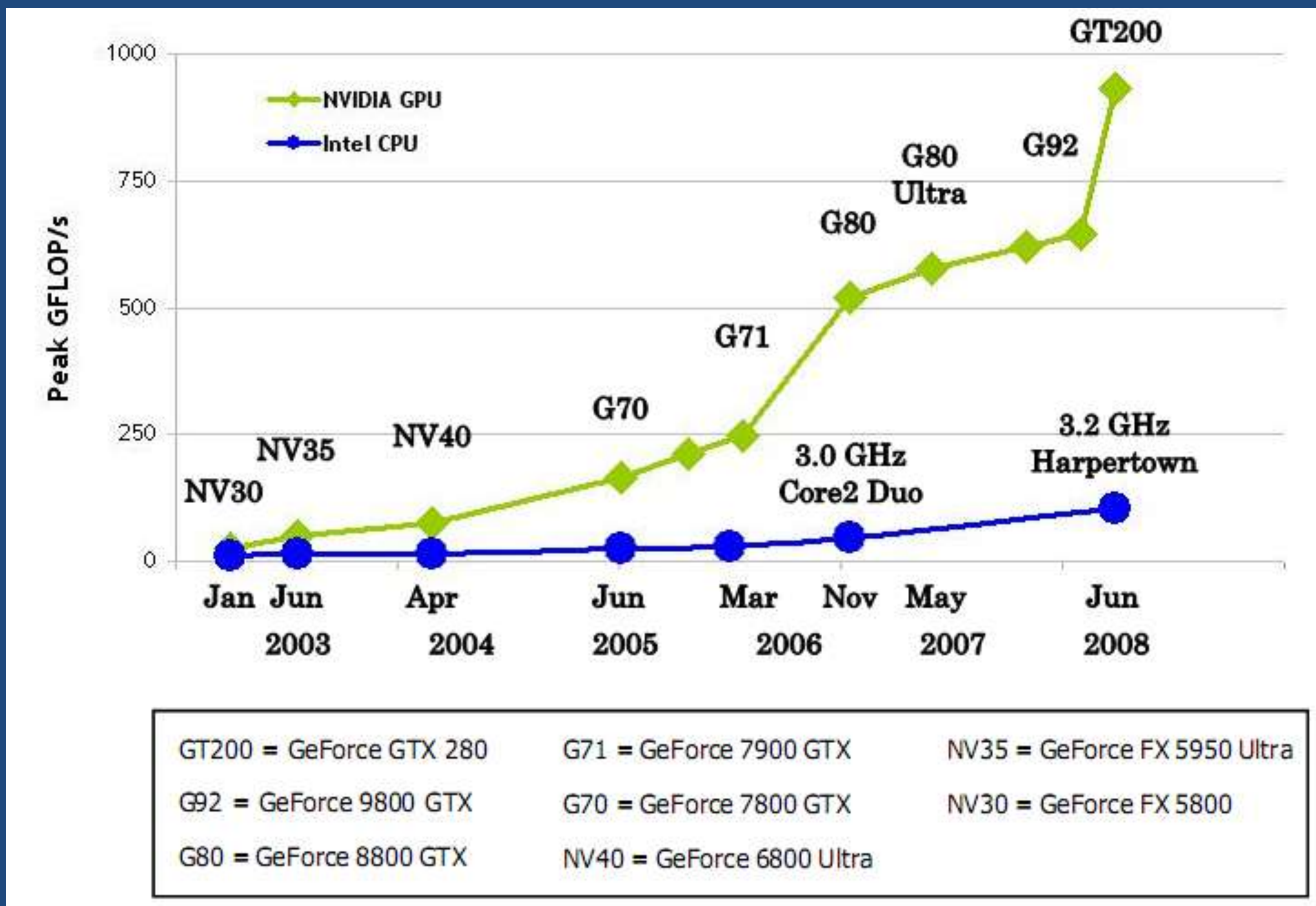
GPU Hardware - Cores

Card	No. processors
GeForce 9800 GTX	128
GeForce 9300M GS	8
GeForce 9800M GTX	112
GeForce GTX 295 (2x285)	2x240
Tesla S1075 (GPU Computing Server)	960
Radeon HD 4890	80
Mobility Radeon HD 5870	160
Radeon HD 5970	2x320

Reasons to use GPGPU

- Power (more info at the next slide)
- Costs (equipment)
- Costs (energy – supercomputers, e.g. Radeon HD 5870 has 14.47 GFLOPS/W and Core 2 Extreme QX9775 has 0.34 GFLOPS/W)
- Memory access (approx. a rank faster for GPU)

GFLOPS



Comparison from *NVIDIA CUDA Programming Guide*

Power comparison

Processing unit (or device)	GFLOPs FP64
Intel Core 2 Duo E8600	26.64
Intel Core 2 Duo P7350 (mobile)	16
Intel Core 2 Quad Q9650	48
Intel Core 2 Extreme QX9775	51.20
NVIDIA GeForce 9300M GS (mobile)	34
NVIDIA GeForce GTX 380	870 (~1700 FP32)
AMD Athlon X2 7750 BE (dual-core)	17
AMD Phenon II X4 940 (quad-core)	44
AMD Radeon HD 5970	928 (~2700 FP32)
Microsoft Xbox 360	115 CPU/240 GPU
Sony PS3	204 CPU / 900 GPU
SPARC64 VIIIfx Venus	128

Various sources i.e. official materials, articles, benchmarks submitted by users.
It is only as a approx. comparison.

GPU Hardware - NVIDIA

GeForce GTX 295
Tesla S1070
Quadro FX 5800
GeForce GTX 285
Tesla C1060
Quadro FX 5600
GeForce GTX 285 for Mac
Tesla C870
Quadro FX 4800
GeForce GTX 280
Tesla D870
Quadro FX 4800 for Mac
GeForce GTX 275
Tesla S870
Quadro FX 4700 X2
GeForce GTX 260
Quadro FX 4600
GeForce GTS 250
Quadro FX 3800
GeForce GTS 240
Quadro FX 3700
GeForce GT 240
Quadro FX 1800
GeForce GT 220
Quadro FX 1700
GeForce 210
Quadro FX 580

GeForce GTS 150
Quadro FX 570
GeForce GT 130
Quadro FX 470
GeForce GT 120
Quadro FX 380
GeForce G100
Quadro FX 380 LP
GeForce 9800
GX2Quadro FX 370
GeForce 9800
GTX+Quadro FX 370 Low
Profile
GeForce 9800
GTXQuadro CX
GeForce 9800
GTQuadro NVS 450
GeForce 9600
GSOQuadro NVS 420
GeForce 9600
GTQuadro NVS 295
GeForce 9500
GTQuadro NVS 290
GeForce 9400GT
Quadro Plex 2100 D4
GeForce 8800
UltraQuadro Plex 2200 D2

GeForce 8800
GTXQuadro Plex 2100 S4
GeForce 8800
GTSQuadro Plex 1000
Model IV
GeForce 8800 GT
GeForce 8800 GS
GeForce 8600 GTS
GeForce 8600 GT
GeForce 8500 GT
GeForce 8400 GS
GeForce 9400 mGPU
GeForce 9300 mGPU
GeForce 8300 mGPU
GeForce 8200 mGPU
GeForce 8100 mGPU
GeForce GTS 360M
Quadro FX 3800M
GeForce GTS 350M
Quadro FX 3700M
GeForce GT 335M
Quadro FX 3600M
GeForce GT 330M
ION Quadro FX 2800M
GeForce GT 325M
Quadro FX 2700M
GeForce 310M

Quadro FX 1800M
GeForce 305M
Quadro FX 1600M
GeForce GTX 285M
Quadro FX 880M
GeForce GTX 280M
Quadro FX 770M
GeForce GTX 260M
Quadro FX 570M
GeForce GTS 260M
Quadro FX 380M
GeForce GTS 250M
Quadro FX 370M
GeForce GTS 160M
Quadro FX 360M
GeForce GTS 150M
Quadro NVS 320M
GeForce GT 240M
Quadro NVS 160M
GeForce GT 230M
Quadro NVS 150M
GeForce GT 130M
Quadro NVS 140M
GeForce G210M
Quadro NVS 135M
GeForce G110M
Quadro NVS 130M

GeForce G105M
GeForce G102M
GeForce 9800M GTX
GeForce 9800M GT
GeForce 9800M GTS
GeForce 9800M GS
GeForce 9700M GTS
GeForce 9700M GT
GeForce 9650M GS
GeForce 9600M GT
GeForce 9600M GS
GeForce 9500M GS
GeForce 9500M G
GeForce 9400M G
GeForce 9300M GS
GeForce 9300M G
GeForce 9200M GS
GeForce 9100M G
GeForce 8800M GTS
GeForce 8700M GT
GeForce 8600M GT
GeForce 8600M GS
GeForce 8400M GT
GeForce 8400M GS

GPU Hardware - ATI

ATI Radeon™ HD
5970
5870
5850
5770
5750
48902
4870 X22
48702
4850 X22
48502
48302
47702
46702
46502
45502
43502

ATI FirePro™
V87502
V87002
V77502
V57002
V37502

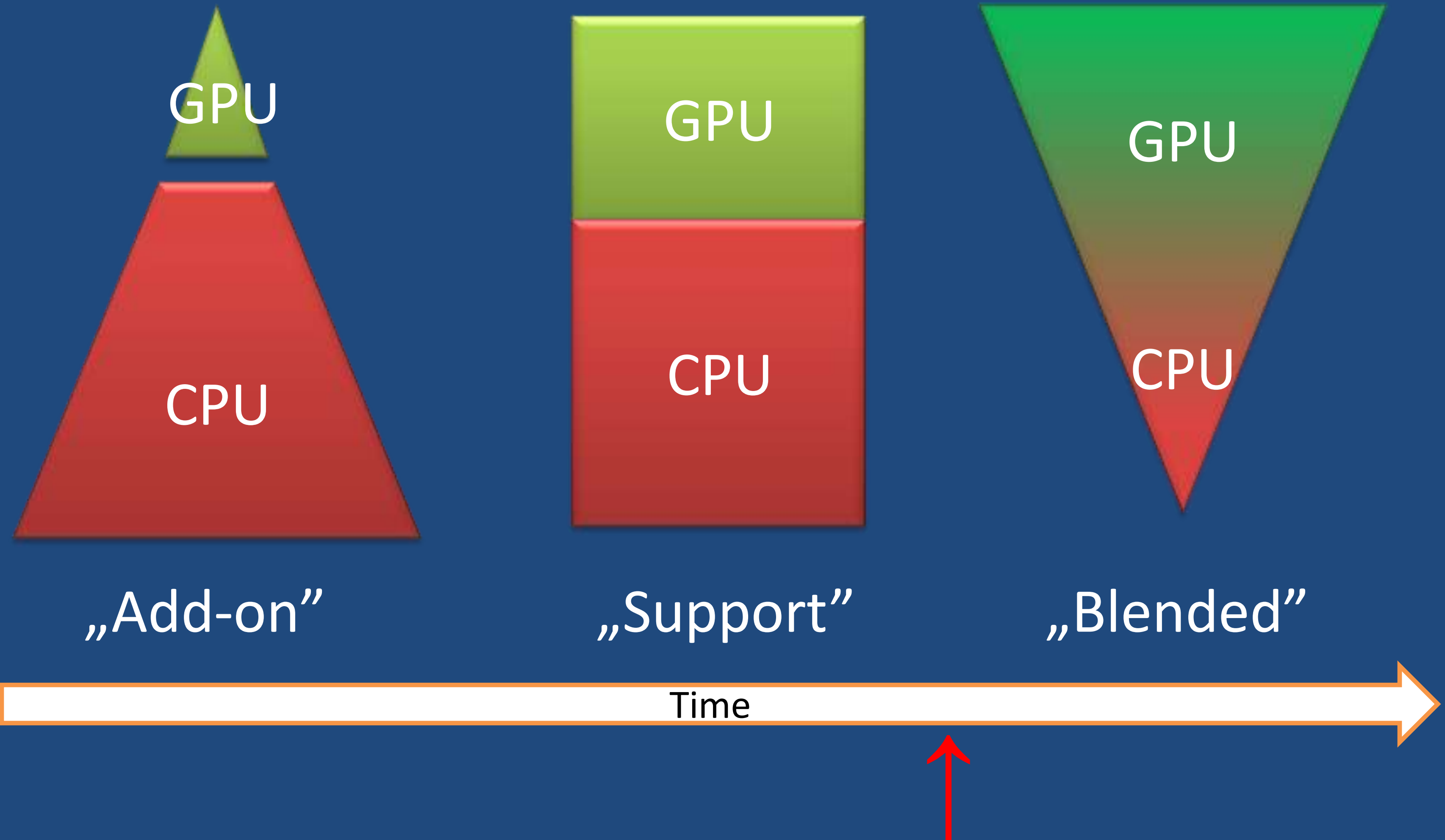
AMD FireStream™
92702
92502

ATI Mobility FirePro™
M77402

ATI Radeon™ Embedded
E4690 Discrete GPU2

ATI Mobility Radeon™ HD
48702
48602
4850X22
48502
48302
46702
46502
4500 Series2
4300 Series2

CPU/GPU tendency



GPGPU Technologies

Techologies

Let's take a look at:

- AMD/ATI Stream
- DirectX11 DirectCompute
- NVIDIA CUDA
- OpenCL

Currently CUDA and Stream support various technologies thus we will limit their analysis only to the original ones.

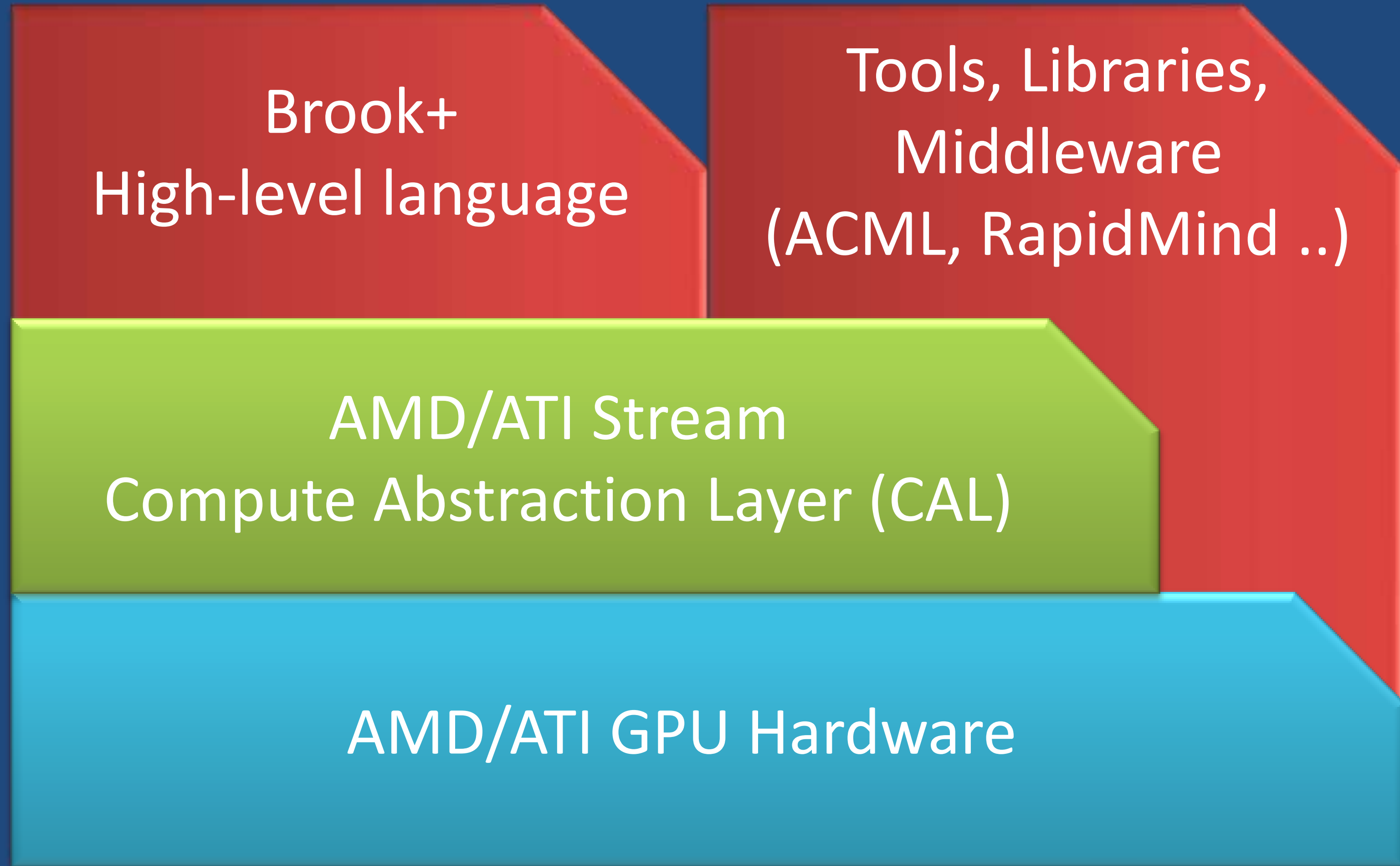
AMD Stream

Released by ATI in December 2007 as *Close-to-the-Metal* (beta version).

Uses *Brook+* language – Brook optimized for AMD hardware.

In later versions was upgraded to use OpenCL and DirectCompute.

AMD Stream



AMD Stream – Brook+

Brooke was an extension to the C language, designed to be used in stream programming (at Stanford University).

Brooke+ was implemented by AMD and build over AMD’s compute abstraction layer and enhanced to use AMD’s hardware.

AMD Stream – Brook+

```
kernel void sum(float a<>, float b<>, out float c<>
{
      c = a + b;
}

int main(int argc, char** argv)
{
      int i, j;
      float a<10, 10>;
      float b<10, 10>;
      float c<10, 10>;
      float input_a[10][10];
      float input_b[10][10];
      float input_c[10][10];
      for(i=0; i<10; i++) {
           for(j=0; j<10; j++) {
                input_a[i][j] = (float) i;
                input_b[i][j] = (float) j;
           }
      }
      streamRead(a, input_a);
      streamRead(b, input_b);
      sum(a, b, c);
      streamWrite(c, input_c);
      ...
}
```

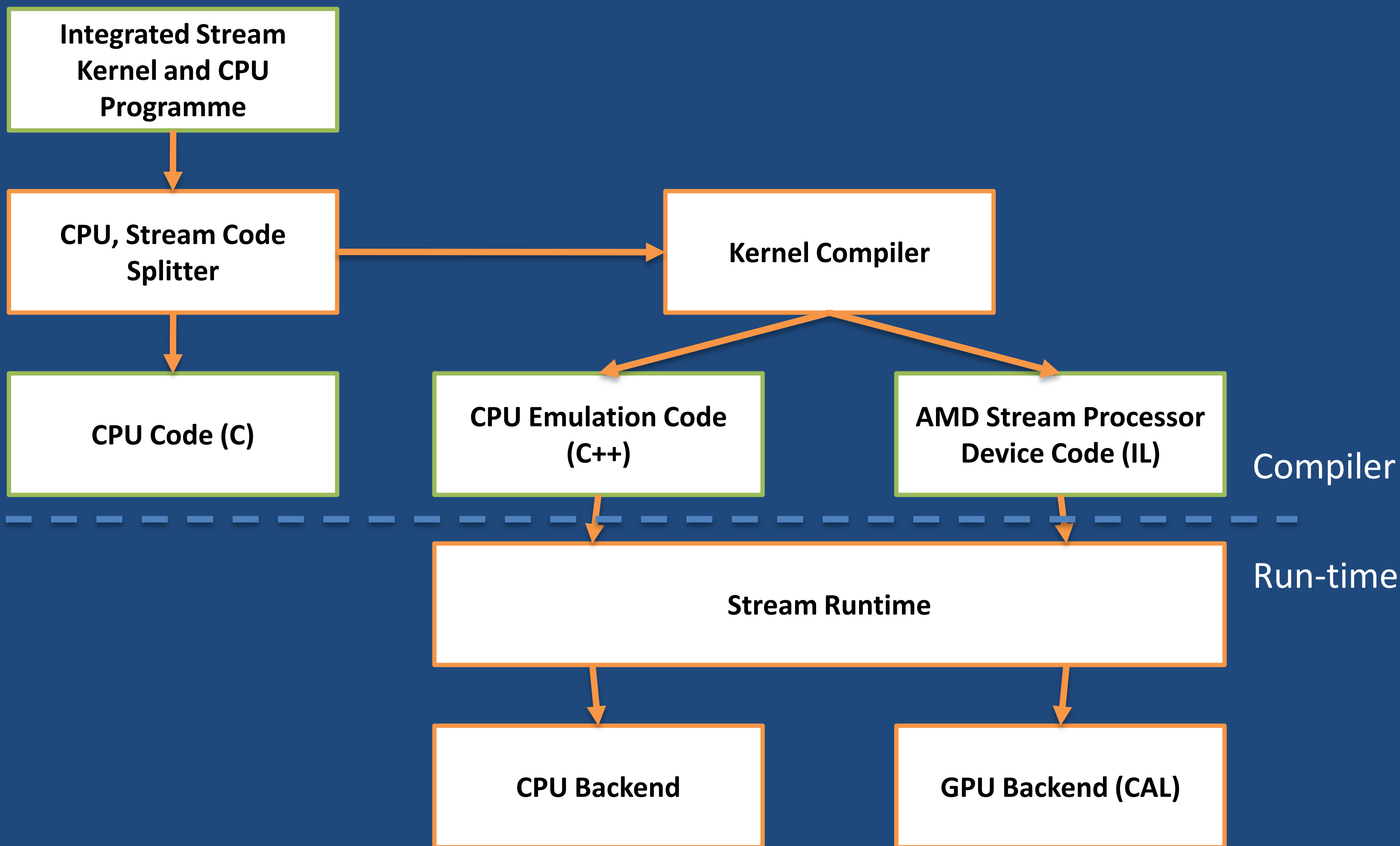
Example comes from AMD-Brookplus Manual

Kernels – operate on stream elements

Streams – collection of data (vectors) to be operated on in parallel

Brook+ access functions

AMD Stream – Tools



AMD Stream – Pros&Cons

+OpenCL was supported by AMD before NVIDIA

-Non-portable technology

-Before AMD introduced OpenCL was CUDA follower

-Less popular than CUDA

DirectX 11 DirectCompute

Microsoft's answer to GPGPU.

Still in the development (drivers, learning resources, tools ….).

A part of DirectX technology that is very popular in computer games industry.

DirectX 11 DirectCompute

DirectX is a winner in PC gaming branch but not in graphical software branch (OpenGL).

Is a step back – causes the software not to be portable (similar to OpenGL/DirectX struggle).

Limited to Windows OS – not well choice for the engineers or scientists.

DirectX 11 DirectCompute

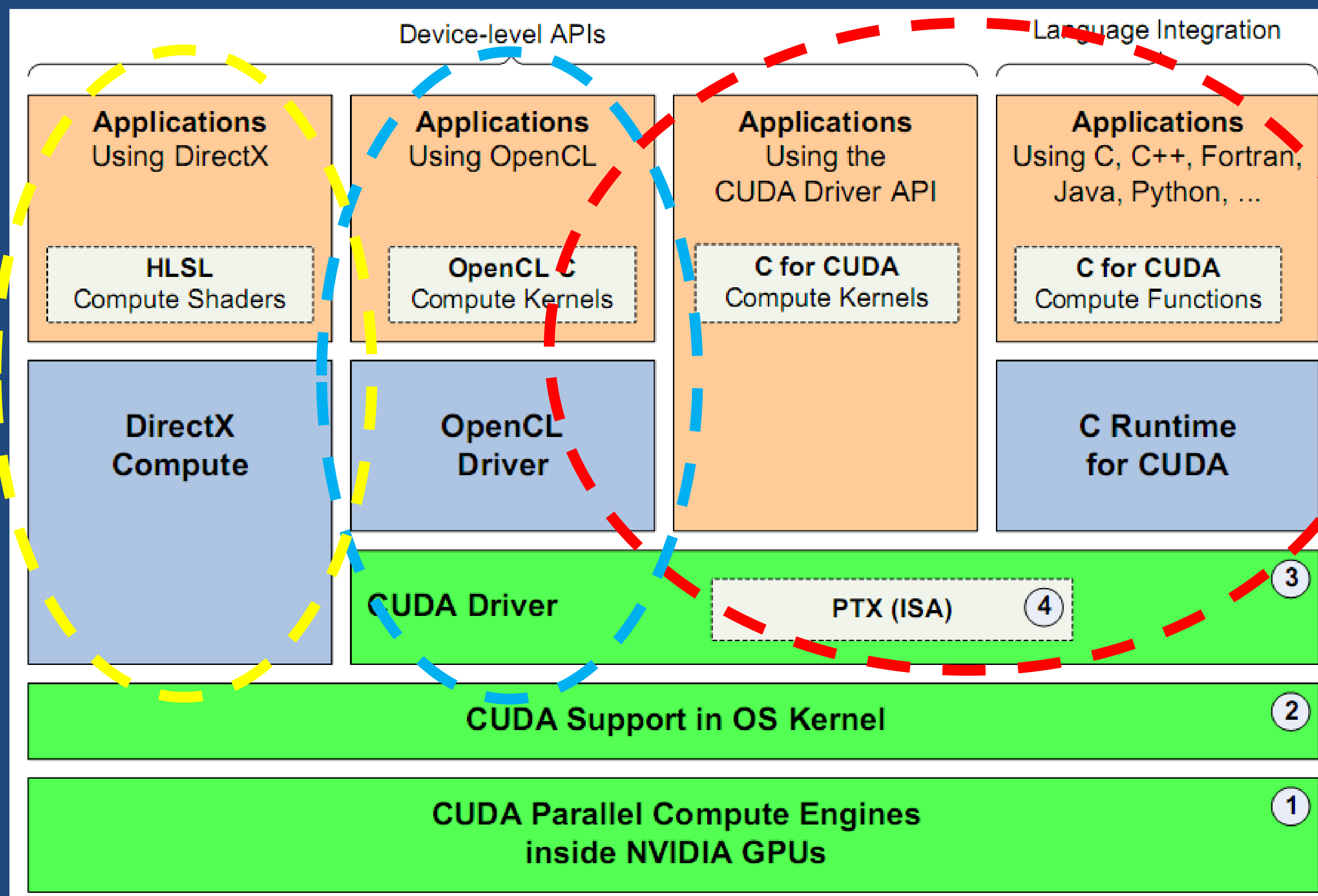
```
//--------------------------------------------------------------------------------------
// File: BasicCompute11.hlsl
//
// This file contains the Compute Shader to perform array A + array B
//
// Copyright (c) Microsoft Corporation. All rights reserved.
//--------------------------------------------------------------------------------------

struct BufType
{
   int i;
   float f;
};

StructuredBuffer<BufType> Buffer0 : register(t0);
StructuredBuffer<BufType> Buffer1 : register(t1);
RWStructuredBuffer<BufType> BufferOut : register(u0);

[numthreads(1, 1, 1)]
void CSMain( uint3 DTid : SV_DispatchThreadID )
{
   BufferOut[DTid.x].i = Buffer0[DTid.x].i + Buffer1[DTid.x].i;
   BufferOut[DTid.x].f = Buffer0[DTid.x].f + Buffer1[DTid.x].f;
}
```

CUDA - History

Public announcement was made in November 2006, Beta was released in February 2007 and full version in June 2007.

Currently CUDA Toolkit is in 3 version (beta).

CUDA - Architecture



Source: NVIDIA CUDA Architecture

C for CUDA

C for CUDA is an extension of C language that allows programmer to target portions of the source code for execution of device.

E.g. functions __global__ and __device__ cannot be recurrent (a it is possible in standard C).

CUDA - Example

```
__global__ void vecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    // Kernel invocation
    vecAdd<<<1, N>>>(A, B, C);
}
```

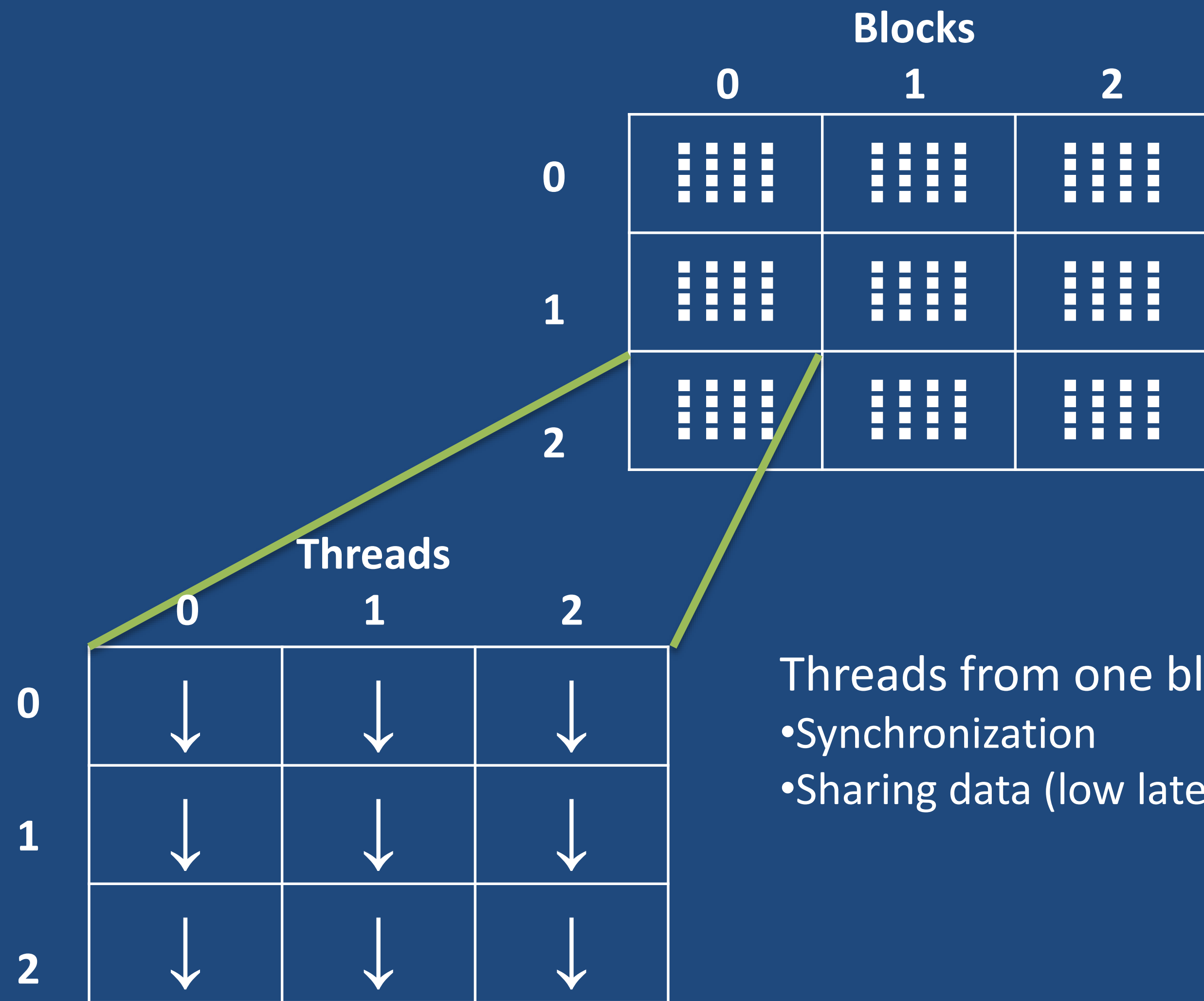
Example comes from *NVIDIA CUDA Programming Guide*

CUDA - Example

```
__global__ void matAdd(float A[N][N], float B[N][N], float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // Kernel invocation
    dim3 dimBlock(N, N);
    matAdd<<<1, dimBlock>>>(A, B, C);
}
```

Example comes from *NVIDIA CUDA Programming Guide*

CUDA – Programming schema



Threads from one block can cooperate by:

•Synchronization

•Sharing data (low latency shared memory)

CUDA - Example

```
__global__ void matAdd(float A[N][N], float B[N][N], float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // Kernel invocation
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N + dimBlock.x – 1) / dimBlock.x,
                 (N + dimBlock.y – 1) / dimBlock.y);
    matAdd<<<dimGrid, dimBlock>>>(A, B, C);
}
```

Example comes from *NVIDIA CUDA Programming Guide*

CUDA – Pros&Cons

+ Uses a variant of C-language

+ Very popular (a privilege of the first solution)

- Single precision FP32 – only cards > GTX 260 have FP64

- Available only on NVIDIA cards

OpenCL

OpenCL is framework managed by consortium Khronos Group. Its development was started by Apple. The consortium includes AMD, IBM, NVIDIA and Intel.

The technical specification was publicly released on 8th December 2008.

OpenCL

OpenCL specification was implemented on various OSs – Windows, MacOS, Linux as well as on various hardware i.e. AMD and NVIDIA.

It is a portable solution (similarly to OpenGL or OpenAL). Vendors must provide Toolkits supporting OpenCL on particular hardware&OS.

OpenCL

This technology was based on C language (C99 variant). Both leaders on GPU market i.e. NVIDIA and AMD currently offers OpenCL support in their GPU toolkits.

It obeys IEEE 754-2008 floating point requirements (CUDA has some differences).

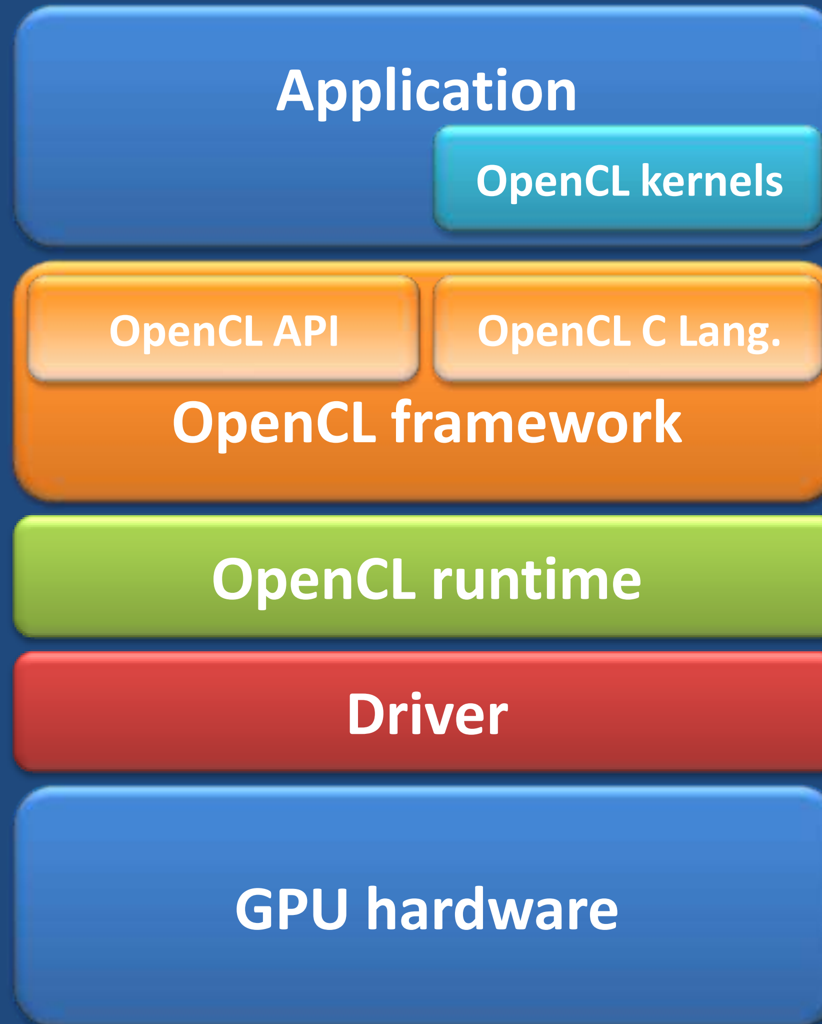
OpenCL

Its idea is to use all computational resources – CPSs, GPUs and other processors.

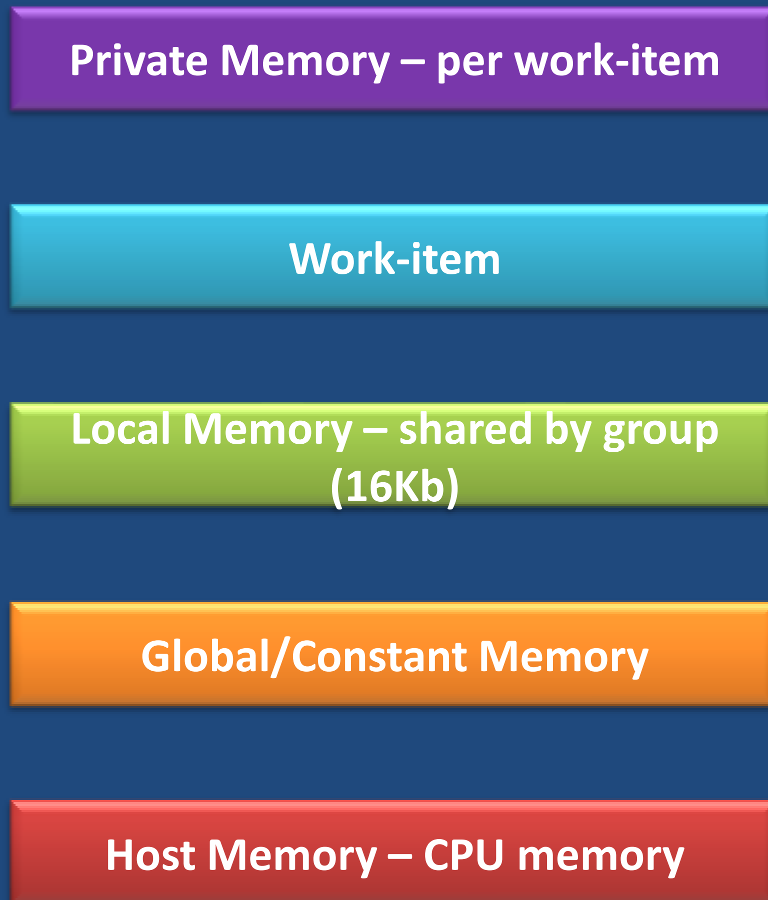
Has desktop and handheld profiles.

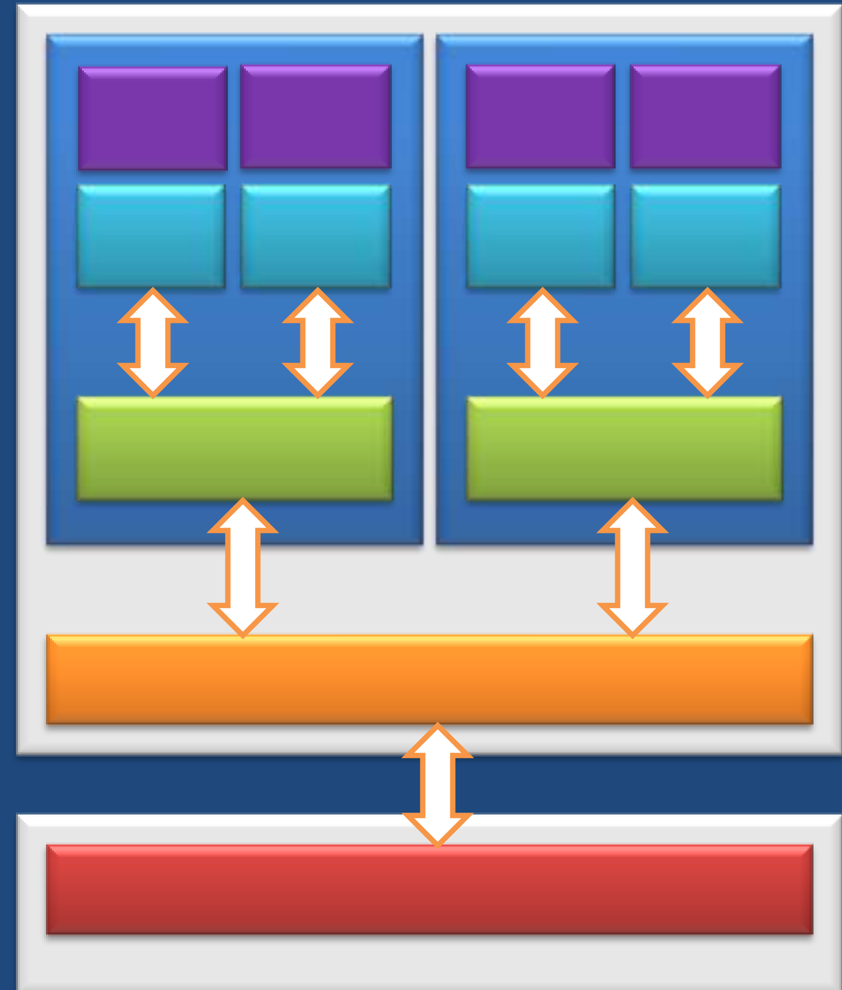
Processing elements are executing code as SIMD and SPMD (Single Process/Program Multiple Data) elements.

OpenCL - Architecture



OpenCL – Memory Model





OpenCL – PyOpenCL Example

```
from numpy import *
import opencl

# CPU vectors allocation and initialization - Classic NumPy
host_vec_1 = array([37,50,54,50,56,12,37,45,77,81,92,56,-22,-4], dtype='int8')
host_vec_2 = array([35,51,54,58,55,32,-5,42,34,33,16,44, 55,14], dtype='int8')
host_vec_out = ndarray(host_vec_1.shape, dtype='int8')

# OpenCL C source
opencl_source = '''
__kernel void
vector_add (__global char *c, __global char *a, __global char *b)
{
  // Index of the elements to add
  unsigned int n = get_global_id(0);
  // Sum the nth element of vectors a and b and store in c
  c[n] = a[n] + b[n];
}
'''

# Compile the code and exec the kernel
prog = opencl.Program(opencl_source)
prog.vector_add(host_vec_out, host_vec_1, host_vec_2)

# Display the result
print ''.join([chr(c) for c in host_vec_out])
```

OpenCL – PyOpenCL Example

```
# Allocate GPU memory
gpu_vec_1 = opencl.Buffer(host_vec_1)
gpu_vec_2 = opencl.Buffer(host_vec_2)
gpu_vec_out = opencl.Buffer(host_vec_out)

# Exec the kernel
prog.vector_add(gpu_vec_out, gpu_vec_1, gpu_vec_2)

# Fetch back results
gpu_vec_out.read(host_vector_out)


gpu_vec_1 = opencl.Buffer(host_vector_1)


gpu_vec_out.read(host_vector_out)
```

OpenCL – Pros&Cons

+ Portable

+ Access to all compute resources

+ Rich set of built-in functions

+ Open standard

-(?) Newer than CUDA/Stream

Programming languages

Programming GPU will not force you to write the all parts of applications in C/C++.

C for writing GPU part is a must but CPU part could be developed in many other languages e.g. Python, Fortran, Java, .NET ….

Steam/CUDA/DC/OpenCL

Steam & CUDA & DirectCompute are unportable

OpenCL:

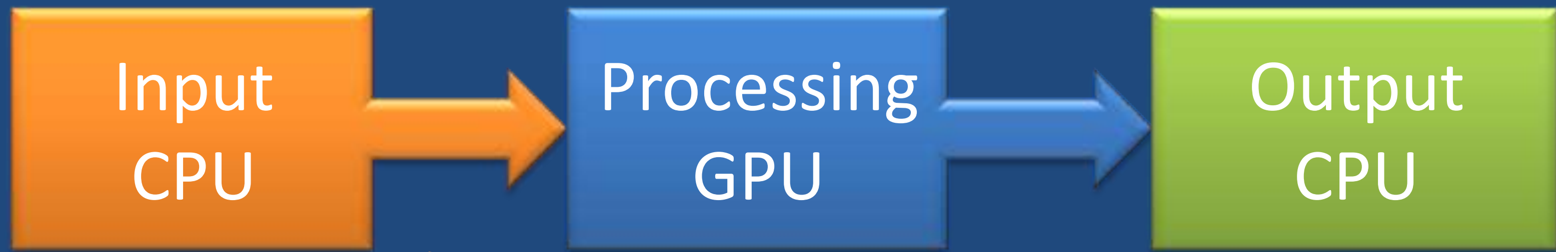
•vendor independent (it Toolkit was provided),
•programmer could focus on an universal problem solving not a specific implementation,
•reduces implementation costs,
•reduces implementation time,

GPGPU-enabled OSs

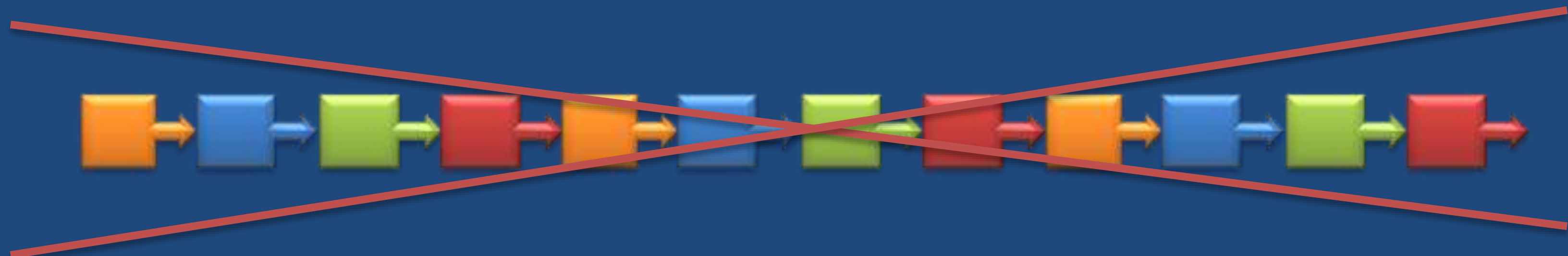
The first GPGPU-enabled OS was MacOS X *Snow Leopard* (10.6). It uses GPGPU to manipulate media e.g. in video de/compression.

We shall expect future OSs to more efficiently manage the processing units (both GPU and CPU cores). 2, 4 or X cores are more marketing tags.

GPGPU performance



Total time of execution



GPGPU performance

Multiply threads

Do as much as possible in each thread – do not move work to the next thread

Watch out for costly memory operations

Use large data sets

GPGPU Software

According to the information given by AMD,

Adobe uses GPGPU in:

- Adobe uses GPGPU in: Acrobat Reader (when working with graphically rich high resolution PDFs)
- Photoshop CS4 Extended (accelerated image and 3D model previewing and manipulations)

while Microsoft uses GPGPU in:

- PowerPoint 2007 (acceleration of slideshow playback)
- Silverlight

What about ...

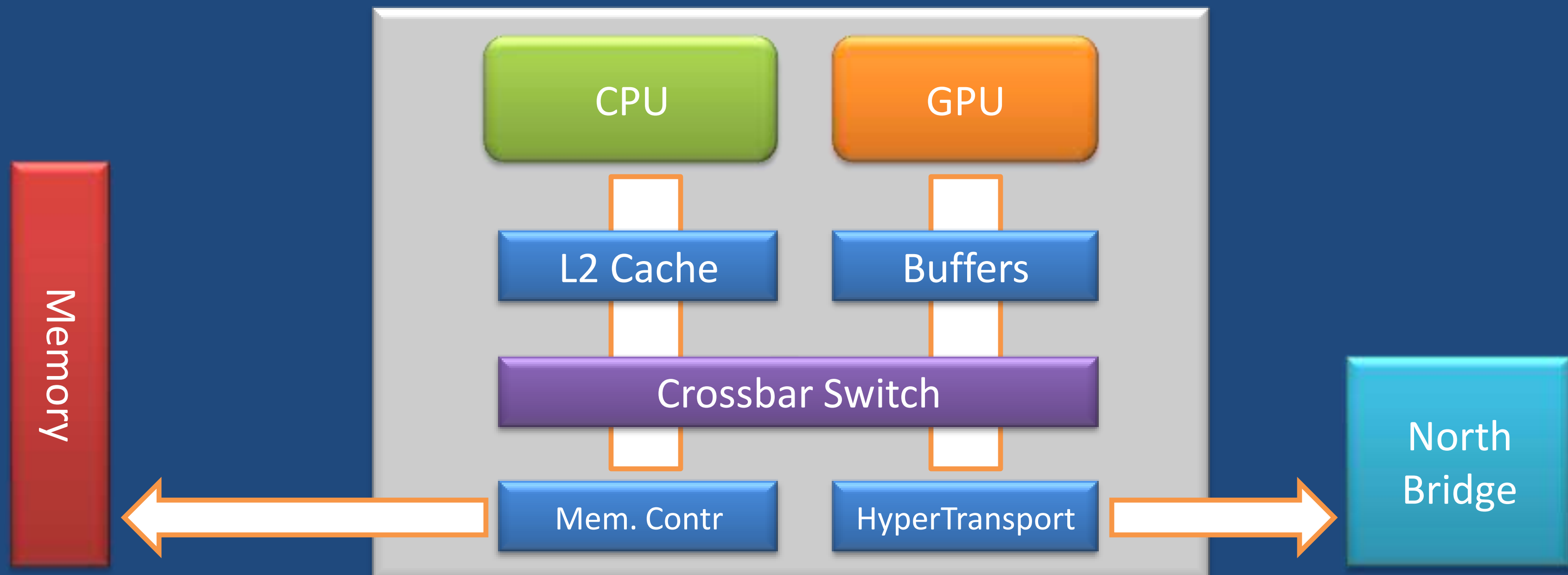
... multi-core processors?

Currently CPUs have 2 (or 4) cores - *Teraflops Research Chip* set a record i.e. it has 80 cores.

Multi-core CPU are less powerful but more flexible. Do not expect GPUs to replace CPUs.

AMD Fusion – CPU/GPU Hybrid

AMD plans to introduce the next generation of microprocessors (called Fusion) in 2011.



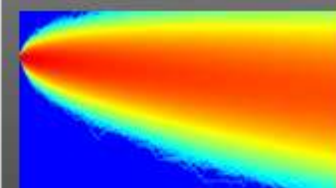
GPGPU Speedup

LATEST CUDA NEWS New Paper on CUDA Zone: GPU-Based Visualization of Billion Point Cosmological Simulations

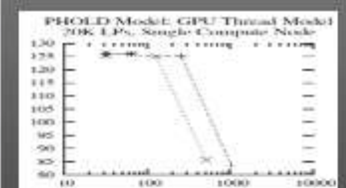
GPU-based Acceleration of the Genetic Algorithm 2600 x

Computer Generated Hologram on GPU - Simple color electroholography 1500 x

Fast Total Variation for Computer Vision 1000 x



Accelerating numerical solution of Stochastic Differential Equations wit 675 x



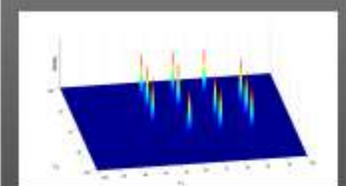
AN APPROACH FOR THE EFFECTIVE UTILIZATION OF GP-GPUS IN PARALLEL 539 x

Real Time Elimination of Undersampling Artifacts in CE MRA using Variational 2300 x

Optimization of FTLE Calulation 1000 x

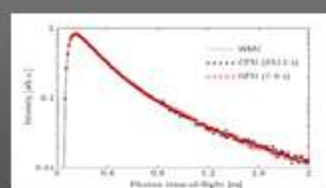
Fast and Exact Solution of Total Variation Models on the GPU 1000 x

TeraChem 650 x

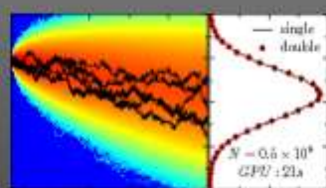


Massively Parallel Population-Based Monte Carlo Methods 500 x

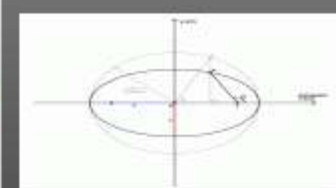
Implementation of a Lattice-Boltzmann method for numerical fluid mechani 1840 x



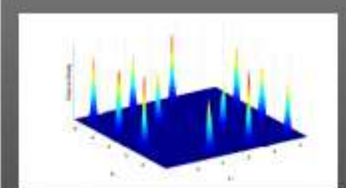
Parallel computing with graphics processing units for high-speed Monte Carlo 1000 x



Stochastic Differential Equations with CUDA 675 x



Parallel Algorithm for Solving Kepler's Equation on Graphics Processing Units: Applica 600 x



On the utility of graphics cards to perform massively parallel simulation of 500 x

CUDA Portfolio

GPU-based Acceleration of the Genetic Algorithm

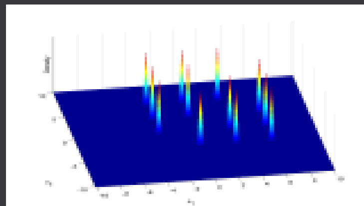
Genetic algorithm (GA) is a stochastic optimization method inspired by nature evolution. Because of their parallel nature, they have been parallelized many times. Graphic Processing Units (GPU) were originally targeted for rasterization of graphics primitives. Today GPUs are more likely fast multi-core processors capable of performing complex mathematical tasks. There are many ways how to exploit GPUs potential for general purpose computation (GPGPU). One option is to employ Compute Unified Device Architecture (CUDA) framework.

Author(s)	Petr Pospichal / Jiri Jaros
Organization Type	Academia
Organization	Brno University of Technology, Bozetechova 2
Software License	

Application Type	Numerics / Science
Speed Up	2600 x
Date Released	12/31/2008
Available Content	

Embed Code <div><script

CUDA Portfolio



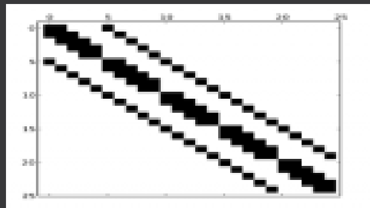
Massively Parallel Population-Based Monte Carlo Methods

Implementation of population-based MCMC and a sequential Monte Carlo sampler for inference in a Gaussian mixture model and a particle filter for a factor stochastic volatility state-space model.

Author(s)	Anthony Lee / Christopher Yau / Michael B. G..	Application Type	Statistics
Organization Type	Academia	Speed Up	500 x
Organization	University of Oxford	Date Released	05/14/2009
Software License	Open source	Available Content	Application / Paper / Code

Embed Code <div><script

CUDA Portfolio



Sparse Matrix-Vector Multiplication Toolkit for Graphics Processing Units

Sparse Matrix-Vector Multiplication Toolkit for Graphics Processing Units (SpMV4GPU) is a library optimized for NVIDIA Graphics Processing Units (GPUs). The GPU is fast emerging as the ideal architecture to use as an accelerator in a heterogenous computing environment. Modern GPUs are designed not only for accelerating traditional graphics kernels, but also for general-purpose computationally intensive kernels. The state-of-the art GPUs exhibit very high computational capabilities at a reasonable price. These GPUs also support high-level parallel programming models, for example, NVIDIA's Common Unified Device Architecture (CUDA) or Brook+ from

Author(s)	Rajesh Bordawekar	Application Type	Computational Fluid Dynamics / Electronic D..
Organization Type	Research	Speed Up	10 x
Organization	IBM Research	Date Released	04/21/2009
Software License	Open source	Available Content	Code

Embed Code <div><script

Bloomberg Case study

Bloomberg calculates pricing for 1.3 million securities. They've parallelized models over x86 Linux machines. In 2005 they released more accurate model which was more computationally expensive. In 2008 the task size forced them to rescale the hardware to calculate prices *on time* –from 800 cores to 8000 core (approx. 1000 servers).

Bloomberg Case study

But they've come with an idea – use GPGPU computing.

Using NVIDIA Tesla GPUs they managed to build sufficient system only with 48 machines (each with a Tesla card).

The x86 servers gather the data and prepare problems to be parallelised while 90% of work is being done by GPUs.

According to Bloomberg they have achieved 800% performance increase.

Scientific applications

Scientific applications

The potential scienfific applications of GPGPU are (but not limited to):

- Optimisation
- Attributes processing
- Filtering
- Classification
- Monte Carlo

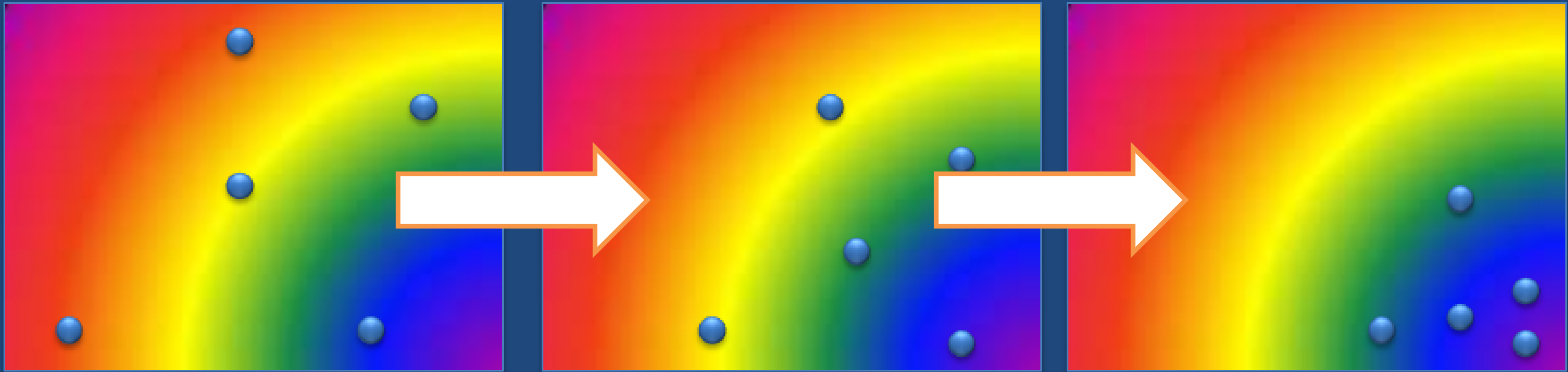
Scientific applications

Optimisation:

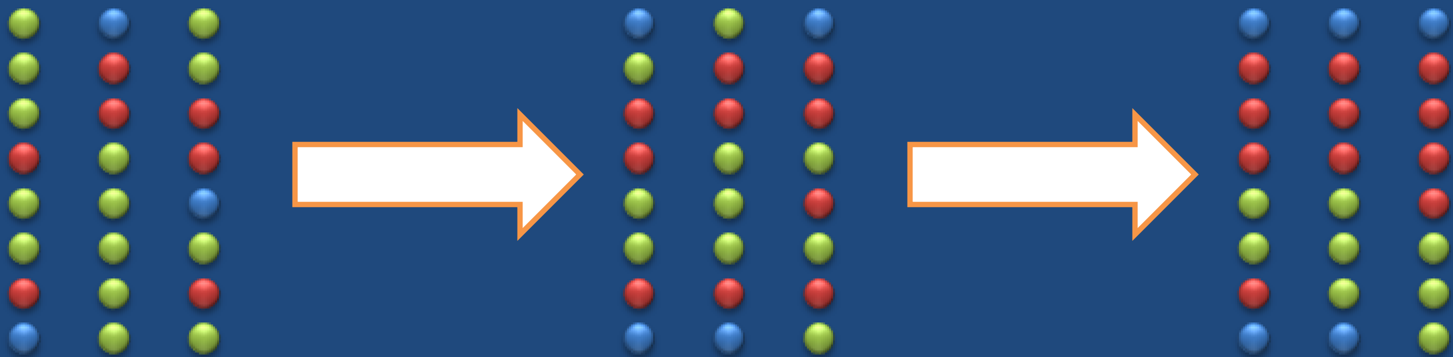
- Genetic algorithms
- Ant colonies
- Particle Swarm Optimisation

Scientific applications

Particla Swarm Optimisation



Genetic Algorithms



Scientific applications

Optimal filtering:

- Lots of matrix calculations
- Complexity is $O(N^3)$ for new filters (but more accurace) or $O(N^2)$ for EKF or SR-versions
- In Particle Filters many particles are passed via system's model

Scientific applications

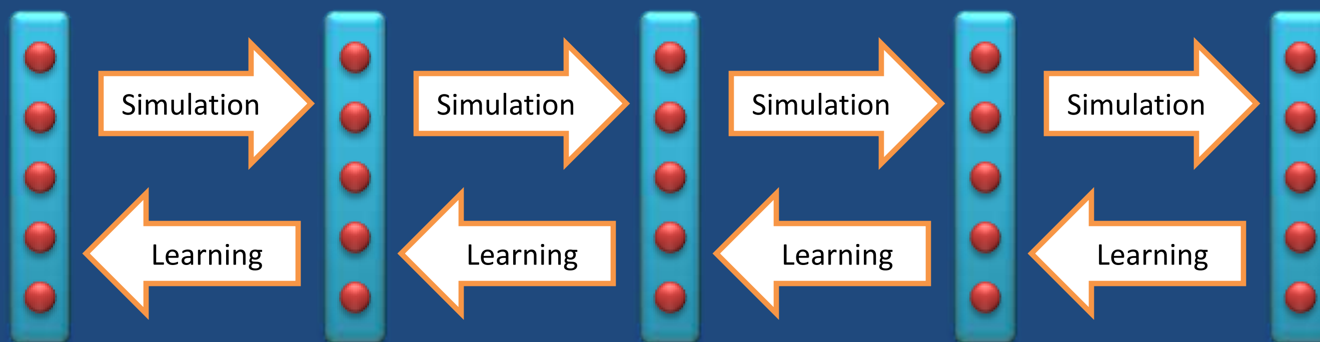
Neural computing:

- Each neuron is a simple processing mechanism
- The net contains tens/hundreds/tousands of them
- They calculations are indepentend (in each layer)

Scientific applications

Neural networks → a lot of simple processing elements = „*That is what ~~Tiggers~~ GPUs do best*”.

The learning for the most of networks contains synchronised parts.



CNL

We (WPUT) started experiments over GPU-based neural networks – CUDA Neural Library http://code.google.com/p/cnl/

The project was started when CUDA was only technology offered by NVIDIA. It is more a test suite than production ready library.

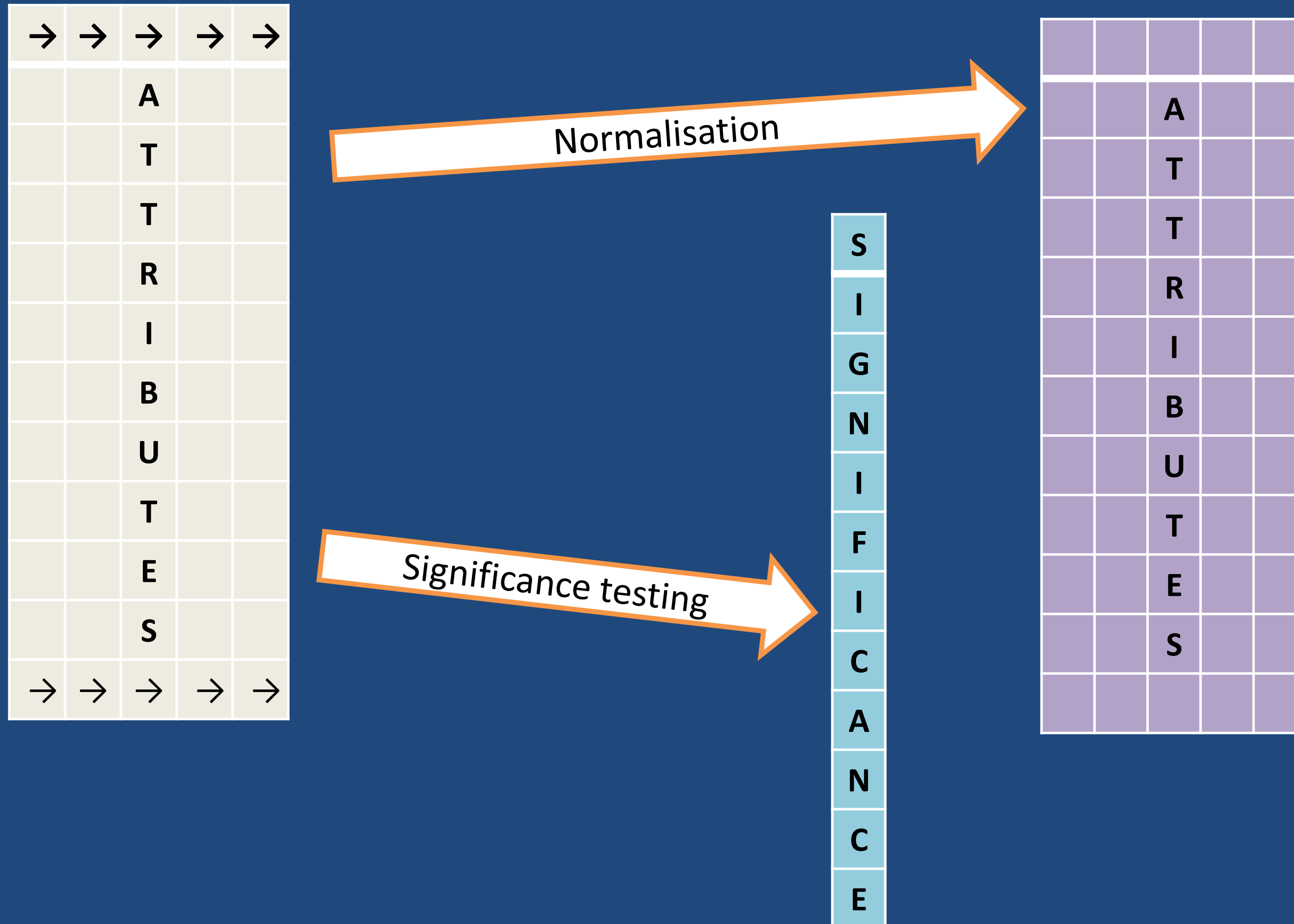
OpenCL is waiting ....

Scientific applications

Attributes processing:

- Selection
- Error removal
- Gap removal
- Transformation (e.g. normalisation)

Scientific applications



Scientific applications

Monte Carlo method:

1. Define a domain of inputs
2. Randomly generate inputs using probability distribution
3. Perform computation using inputs
4. Using individual computation aggregate the results into the final results

If you look further …

- GPGPU.org - http://www.gpgpu.org
- Microsoft DirectX 11 Samples - http://microsoftpdc.com/Sessions/P09-16
- CUDA Zone - http://www.NVIDIA.com/object/cuda_home.html
- OpenCL – http://www.khronos.org/opencl/
- AMD Stream – http://www.amd.com/stream
- PyOpenCL - http://pypi.python.org/pypi/pyopencl

Thank you!
Any questions?